Balancing Cost and Benefit with Tied-Multi Transformers

Raj Dabre **Raphael Rubino** **Atsushi Fujita**
National Institute of Information and Communications Technology
3-5 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0289, Japan
firstname.lastname@nict.go.jp

Abstract

We propose a novel procedure for training multiple Transformers with tied parameters which compresses multiple models into one enabling the dynamic choice of the number of encoder and decoder layers during decoding. In training an encoder-decoder model, typically, the output of the last layer of the N-layer encoder is fed to the M-layer decoder, and the output of the last decoder layer is used to compute loss. Instead, our method computes a single loss consisting of $N \times M$ losses, where each loss is computed from the output of one of the M decoder layers connected to one of the N encoder layers. Such a model subsumes $N \times M$ models with different number of encoder and decoder layers, and can be used for decoding with fewer than the maximum number of encoder and decoder layers. Given our flexible tied model, we also address to a-priori selection of the number of encoder and decoder layers for faster decoding, and explore recurrent stacking of layers and knowledge distillation for model compression. We present a cost-benefit analysis of applying the proposed approaches for neural machine translation and show that they reduce decoding costs while preserving translation quality.

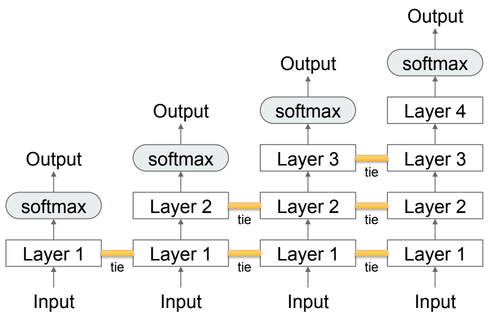
1 Introduction

Neural networks for sequence-to-sequence modeling typically consist of an encoder and a decoder coupled via an attention mechanism. Whereas the very first deep models used stacked recurrent neural networks (RNN) (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015) in the encoder and decoder, the recent Transformer model (Vaswani et al., 2017) constitutes the current state-of-the-art approach, owing to its better context modeling via multi-head self- and cross-attentions.

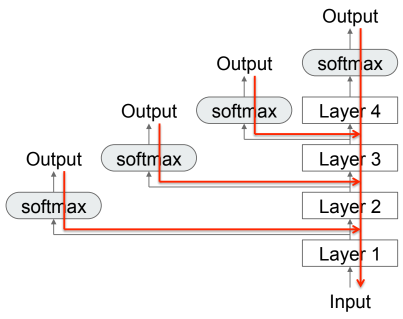
Given an encoder-decoder architecture and its hyper-parameters, such as the number of encoder and decoder layers, vocabulary sizes (in the case of models for texts), and hidden layers, the parameters of the model, i.e., matrices and biases for non-linear transformations, are optimized by iteratively updating them so that the loss for the training data is minimized. The hyper-parameters can also be tuned, for instance, through maximizing the automatic evaluation score on the development data. However, in general, it is highly unlikely (or impossible) that a single optimized model suffices diverse cost-benefit demands at the same time. For instance, in practical low-latency scenarios, one may accept some performance drop for speed. However, a model used with a subset of optimized parameters might perform badly. Also, a single optimized model cannot guarantee the best performance for each individual input. An existing solution for these problems is to train multiple models and host them simultaneously. However, this approach is not very practical, because it requires a large number of resources. We also lack a well-established method for selecting appropriate models for each individual input prior to decoding.

As a more effective solution, we consider training a single model that subsumes multiple models which can be used for decoding with different hyper-parameter settings depending on the input or on the latency requirements. In this paper, we focus on the number of layers as an important hyper-parameter that impacts both speed and quality of decoding, and propose a *multi-layer softmaxing* method, which uses the outputs of all layers during training. Conceptually, as illustrated in Figure 1, this is the same as tying (sharing) the parameters of multiple models with different number of layers and is not specific to particular types of multi-layer neural models.

Despite the generality of our proposed method, in this paper, we focus on encoder-decoder models with N encoder and M decoder layers, and



(a) Multiple tied-layer vanilla models.



(b) Collapsing tied layers into one.

Figure 1: The general concept of multi-layer softmaxing for training multi-layer neural models with an example of a 4-layer model. Figure 1a is a depiction of our idea in the form of multiple vanilla models whose layers are tied together. Figure 1b shows the result of collapsing all tied layers into a single layer. The red lines indicate the flow of gradients and hence the shallowest layer in the stack receives the largest number of updates.

compress $N \times M$ models[1] by updating the model with a total of $N \times M$ losses computed by softmaxing the output of each of the M decoder layers, where it attends to the output of each of the N encoder layers. The number of parameters of the resultant encoder-decoder model is equivalent to that of the most complex subsumed model with N encoder and M decoder layers. Yet, we can now perform faster decoding using a fewer number of layers, given that shallower layers are also directly trained.

To evaluate our proposed method, we take the case study of neural machine translation (NMT) (Cho et al., 2014; Bahdanau et al., 2015), using the Transformer model (Vaswani et al., 2017), and demonstrate that a single model with N encoder and M decoder layers trained by our method can be used for flexibly decoding with fewer than N and M layers without appreciable quality loss. We evaluate our proposed method on WMT18 English-to-German translation task, and give a cost-benefit analysis for translation quality vs. decoding speed.

Given a flexible tied model, for saving decoding time, we then design mechanisms to choose, prior to decoding, the appropriate number of encoder and decoder layers depending on the input. We also focus on compact modeling, where we leverage other orthogonal types of parameter tying approaches. Compact models are faster to decode and will be useful in cases where a-priori layer prediction might be infeasible.

The rest of the paper is organized as follows. Section 2 briefly reviews related work for compressing neural models. Section 3 covers our method that ties multiple models by softmaxing all encoder-decoder layer combinations. Section 4 describes our efforts towards designing and evaluating a mechanism for dynamically selecting encoder-decoder layer combinations prior to decoding. Section 5 describes two orthogonal extensions to our model aiming at further model compression and speeding-up of decoding. The paper ends with Section 6 containing conclusion and future work.

[1]Rather than casting the encoder-decoder model into a single column model with $(N + M)$ layers.

2 Related Work

There are studies that exploit multiple layers simultaneously. Wang et al. (2018) fused hidden representations of multiple layers in order to improve the translation quality. Belinkov et al. (2017) and Dou et al. (2018) attempted to identify which layer can generate useful representations for different natural language processing tasks. Unlike them, we make all layers of the encoder and decoder usable for decoding with any encoder-decoder layer combination. In practical scenarios, we can save significant amounts of time by choosing shallower encoder and decoder layers for inference.

Our method ties the parameters of multiple models, which is orthogonal to the work that ties parameters between layers (Dabre and Fujita, 2019) and/or between the encoder and decoder within a single model (Xia et al., 2019; Dabre and Fujita, 2019). Parameter tying leads to compact models, but they usually suffer from drops in inference quality. In this paper, we counter such drops with knowledge distillation (Hinton et al., 2015; Kim and Rush, 2016; Freitag et al., 2017). This approach utilizes smoothed data or smoothed training signals instead of the actual training data. A model with a large number of parameters and high performance provides smoothed distributions that are then used as labels for training small models instead of one-hot vectors.

As one of the aims in this work is model size reduction, it is related to a growing body of work that addresses the computational requirement reduction. Pruning of pre-trained models (See et al., 2016) makes it possible to discard around 80% of the smallest weights of a model without deterioration in inference quality, given it is re-trained with appropriate hyper-parameters after pruning. Currently, most deep learning implementations use 32-bit floating point representations, but 16-bit floating point representations (Gupta et al., 2015; Ott et al., 2018) or aggressive binarization (Courbariaux et al., 2017) can be alternatives. Compact models are usually faster to decode; studies on quantization (Lin et al., 2016) and average attention networks (Xiong et al., 2018) address this topic.

None of the above work has attempted to combine multi-model parameter tying, knowledge distillation, and dynamic layer selection for obtaining and exploiting highly-compressed and flexible deep neural models.

3 Multi-Layer Softmaxing

3.1 Proposed Method

Consider an N-layer encoder and M-layer decoder model. Let X be the embedded input to the encoder, Y the expected output of the decoder as well as the input to the decoder (for training), and $\hat{Y}$ the output predicted by the decoder. Algorithm 1 shows the pseudo-code for our proposed method. Line 3 represents the process done by the i-th encoder layer, L_i^{enc}, and line 5 does the same for the j-th decoder layer, L_j^{dec}, given the embedded decoder input, dec_0. In simple words, we compute a loss using the output of each of the M decoder layers which in turn is computed using the output of each of the N encoder layers. In line 8, the $N \times M$ losses are aggregated[2] before back-propagation. Henceforth, we will refer to this as the *Tied-Multi model*.

For a comparison, the vanilla model is formulated as follows: $dec_j = L_j^{dec}(dec_{j-1}, enc_N)$, $\hat{Y} = softmax(dec_M)$, and $overall\_loss = cross\_entropy(\hat{Y}, Y)$.

[2]We averaged multiple losses in our experiment, but there are a number of options, such as weighted averaging.

Algorithm 1: Training a tied-multi model

```
1 $enc_0 = X$;
2 for $i$ in 1 to $N$ do
3     $enc_i = L_i^{enc}(enc_{i-1})$;
4     for $j$ in 1 to $M$ do
5         $dec_j = L_j^{dec}(dec_{j-1}, enc_i)$;
6         $\hat{Y} = softmax(dec_j)$;
7         $loss_{i,j} = cross\_entropy(\hat{Y}, Y)$;
8 $overall\_loss = aggregate(loss_{1,1}, \ldots, loss_{N,M})$;
9 Back-propagate using $overall\_loss$;
```

3.2 Experimental Setup

We evaluated the utility of our multi-layer softmaxing method on a neural machine translation task. We experimented with the WMT18 English-to-German (En→De) translation task. We used all the parallel corpora available for WMT18, except ParaCrawl corpus,[3] consisting of 5.58M sentence pairs, as the training data and 2,998 sentences in newstest2018 as test data. The English and German sentences were pre-processed using the `tokenizer.perl` and `truecase.perl` in `Moses`.[4] The true-case models for English and German were trained on 10M sentences randomly extracted from the monolingual data made available for the WMT18 translation task, using the `train-truecaser.perl` in `Moses`.

We evaluated the following two types of models on both translation quality and decoding speed.

Vanilla models: 36 vanilla models with 1 to 6 encoder and 1 to 6 decoder layers, each trained referring only to the last layer for computing loss.

Tied-Multi model: A single tied-multi model with $N = 6$ encoder and $M = 6$ decoder layers, trained by our multi-layer softmaxing.

Our multi-layer softmaxing method was implemented on top of an open-source toolkit of the Transformer model (Vaswani et al., 2017) in the version 1.6 branch of `tensor2tensor`.[5] For training, we used the default model settings corresponding to `transformer_base_single_gpu` in the implementation, except what follows. We used a shared sub-word vocabulary of 32k determined using the internal sub-word segmenter of tensor2tensor. To ensure that each model sees roughly

[3]http://www.statmt.org/wmt18/translation-task.html We excluded ParaCrawl following the instruction on the WMT18 website: "BLEU score dropped by 1.0" for this task.

[4]https://github.com/moses-smt/mosesdecoder

[5]https://github.com/tensorflow/tensor2tensor

	BLEU score												Decoding time (sec)					
	36 vanilla models						Single tied-multi model											
$n\backslash m$	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
1	26.3	30.3	31.9	32.2	32.4	32.9	23.2	28.6	30.5	30.8	31.2	31.5	94.7	101.9	143.4	174.7	215.5	244.5
2	28.6	32.5	33.1	33.3	33.5	33.2	26.5	31.5	33.0	33.6	33.8	34.0	100.5	110.8	153.7	185.6	227.8	253.6
3	29.2	32.6	33.6	34.4	34.3	34.1	27.8	32.5	33.9	34.6	34.7	34.7	102.5	114.2	168.5	194.8	234.0	259.8
4	29.8	33.6	34.3	34.7	34.4	34.5	28.3	33.0	34.3	34.8	34.9	34.9	104.1	105.6	143.9	197.0	219.1	264.6
5	30.7	33.9	34.6	35.5	34.4	35.0	28.6	33.1	34.5	34.8	35.0	35.1	105.1	111.5	156.4	186.0	236.1	268.8
6	30.8	34.0	34.4	**35.7**	35.0	35.0	28.7	33.1	34.6	34.7	34.9	35.0	107.4	113.6	168.1	190.1	229.5	257.9

Table 1: BLEU scores of 36 separately trained vanilla models and our single tied-multi model used with n ($1 \leq n \leq N$) encoder and m ($1 \leq m \leq M$) decoder layers. One set of decoding times is also shown given the fact that vanilla and our tied-multi models have identical shapes when n and m for encoder and decoder layers are specified.

the same number of examples during training,[6] we trained the models for 300k iterations, with 1 GPU for the vanilla models and 2 GPUs with batch size halved for our tied-multi model. We averaged the last 10 checkpoints saved every after 1k updates, decoded the test sentences, fixing a beam size[7] of 4 and length penalty of 0.6, and post-processed the decoded results using the `detokenizer.perl` and `detruecase.perl` in `Moses`.

We evaluated our models using the BLEU metric (Papineni et al., 2002) implemented in sacreBLEU (Post, 2018).[8] We also present the time consumed to translate the test data, which includes times for the model instantiation, loading the checkpoints, sub-word splitting and indexing, decoding, and sub-word de-indexing and merging, whereas times for detokenization are not taken into account.

Note that we did not use any development data for two reasons. First, we train all models for the same number of iterations. Second, we use checkpoint averaging before decoding, which does not require development data unlike early stopping.

3.3 Results

Table 1 summarizes the BLEU scores and the average decoding times[9] over 3 runs of all the models, exhibiting the cost-benefit property of our tied-multi model in comparison with the results of the corresponding 36 vanilla models.

Even though the objective function for the tied-multi model is substantially more complex than the one for the vanilla model, when performing decoding with the 6 encoder and 6 decoder layers, it achieved a BLEU score of 35.0, which is approaching to the best BLEU score of 35.7 given by the vanilla model with 6 encoder and 4 decoder layers. Note that when using a single encoder layer and/or a single decoder layer, the vanilla models gave significantly higher BLEU score than the tied-multi model. However, when the number of layers is increased, there is no significant difference between the two types of models.

Regarding the cost-benefit property of our tied-multi model, two points must be noted:

- BLEU score and decoding time increase only slightly, when we use more encoder layers.
- The bulk of the decoding time is consumed by the decoder, since it works in an auto-regressive manner. We can substantially cut down decoding time by using fewer decoder layers which does lead to sub-optimal translation quality.

One may argue that training a single vanilla model with optimal number of encoder and decoder layers is enough. However, as discussed in Section 1, it is impossible to know a priori which combination is the best for different input sentences. More importantly, a single vanilla model cannot suffice diverse cost-benefit demands and cannot guarantee the best translation for any input (see Section 3.4). Recall that we aim at a flexible model and that all the results in Table 1 have been obtained using a single tied-multi model, albeit using different number of encoder and decoder layers for decoding.

3.4 Analysis and Discussion

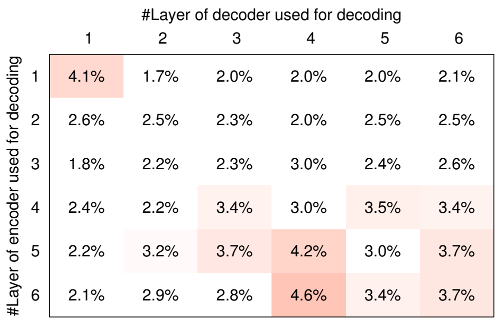
We conducted an analysis from the perspective of training time, model size, and decoding behavior, in comparison with vanilla models.

[6] This might lead to sub-optimal models, such as immature or over-fit ones, so we will examine the convergence in future.

[7] One can realize faster decoding by narrowing down the beam width. This approach is orthogonal to ours and in this paper we do not insist which is superior to the other.

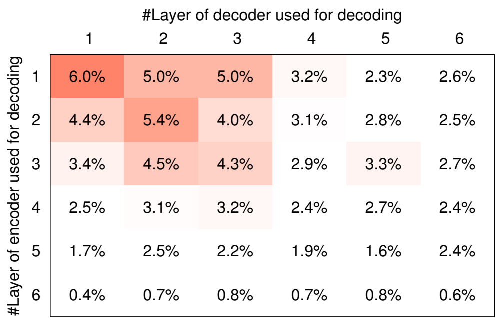
[8] https://github.com/mjpost/sacreBLEU
signature: BLEU+case.mixed+lang.en-de+numrefs.1 +smooth.exp+test.wmt18+tok.13a+version.1.3.7

[9] These numbers will vary depending on machine, model architecture, concurrent processes, implementation, hyper-parameters, etc. For instance, decoding with a larger length penalty produces longer sentences consuming a longer time.



#Layer of encoder used for decoding \ #Layer of decoder used for decoding	1	2	3	4	5	6
1	4.1%	1.7%	2.0%	2.0%	2.0%	2.1%
2	2.6%	2.5%	2.3%	2.0%	2.5%	2.5%
3	1.8%	2.2%	2.3%	3.0%	2.4%	2.6%
4	2.4%	2.2%	3.4%	3.0%	3.5%	3.4%
5	2.2%	3.2%	3.7%	4.2%	3.0%	3.7%
6	2.1%	2.9%	2.8%	4.6%	3.4%	3.7%

(a) 36 vanilla models.



#Layer of encoder used for decoding \ #Layer of decoder used for decoding	1	2	3	4	5	6
1	6.0%	5.0%	5.0%	3.2%	2.3%	2.6%
2	4.4%	5.4%	4.0%	3.1%	2.8%	2.5%
3	3.4%	4.5%	4.3%	2.9%	3.3%	2.7%
4	2.5%	3.1%	3.2%	2.4%	2.7%	2.4%
5	1.7%	2.5%	2.2%	1.9%	1.6%	2.4%
6	0.4%	0.7%	0.8%	0.7%	0.8%	0.6%

(b) Single tied-multi model.

Figure 2: Distribution of oracle translations determined by chrF scores between reference and each of the hypotheses derived from the 36 combinations of encoder and decoder layers (newstest2018, 2,998 sentences).

Training Time: Given that all our models were trained for the same number of iterations, we compared the training times between vanilla and tied-multi models. As a reference, we use the vanilla model with 6 encoder and 6 decoder layers. The total training time for all the 36 vanilla models was 25.5 times[10] that of the reference model. In contrast, the training time for our tied-multi model was about 9.5 times that of the same reference model. This is because training of a tied-multi model can aggressively leverage GPU parallellism for its vast number of computations.

Model Size: The number of parameters of our tied-multi model is exactly the same as the vanilla model with N encoder and M decoder layers. If we train a set of vanilla models with different numbers of encoder and decoder layers, we end up with significantly more parameters. For instance, in case of $N = M = 6$ in our experiment, we have 25.2 times more parameters: a total of 4,607M for the 36 vanilla models against 183M for our tied-multi model. In Section 5, we discuss the possibility of further model compression.

Decoding Behavior: To better understand the nature of our proposed method, we analyzed the distribution of oracle translations within 36 translations generated by each of the vanilla and our tied-multi models. Let (n, m) be an encoder-decoder layer combination of a given model with n encoder and m decoder layers. The oracle layer combination for an input sentence was determined by measuring the quality of the translation derived from each layer combination. We used a reference-based metric, chrF (Popović, 2016), since it has been particularity designed for sentence-level translation evaluation and was shown to have relatively high correlation with human judgment of translation quality at sentence level for the English–German pair (Ma et al., 2018). In cases where multiple combinations have the highest score, we chose the fastest one following the overall trend of decoding time (Table 1). Formally, we considered a combination (n_1, m_1) is faster than another combination (n_2, m_2) if the following holds.

$$\begin{aligned}&(n_1, m_1) < (n_2, m_2)\\&\equiv m_1 < m_2 \vee (m_1 = m_2 \wedge n_1 < n_2).\end{aligned} \quad (1)$$

Figure 2 compares the distributions of oracle layer combinations for the vanilla and our tied-multi models, revealing that the shallower layer combinations in our tied-multi model often generates better translations than deeper ones unlike the vanilla models, despite the lower corpus-level BLEU scores. This sharp bias towards shallower layer combinations suggests the potential reduction of decoding time by dynamically selecting the layer combination per input sentence prior to decoding, ideally without performance drop. We address this task in Section 4.

4 Dynamic Layer Selection

Motivated by the results shown in Figure 2, we tackled an advanced problem: dynamic selection of one layer combination prior to decoding.[11]

4.1 Method

We formalize the encoder-decoder layer combination selection with a supervised learning approach

[10] We measured the collapsed time for a fair comparison, assuming that all vanilla models were trained on a single GPU one after another, even though one may be able to use multiple GPUs to train the 36 vanilla models in parallel.

[11] This is the crucial difference from two post-decoding processes: translation quality estimation (Specia et al., 2010) and n-best-list re-ranking (Kumar and Byrne, 2004).

where the objective is to minimize the following loss function (2).

$$\arg\min_{\theta} \frac{1}{|S|} \sum_{s^i \in S} \mathcal{L}(f(s^i;\theta), t^i_k), \quad (2)$$

where s^i is the i-th input sentence ($1 \leq i \leq |S|$), t^i_k is the translation for s^i derived from the k-th layer combination ($1 \leq k \leq K$) among K possible combinations, where $K = N \times M$ in our case, f is the model with parameters θ, and $\mathcal{L}$ is a loss function. Assuming that the independence of target labels (layer combinations) for a given input sentence allows for ties, the model is able to predict multiple layer combinations for the same input sentence.

We implemented the model f with a multi-head self-attention neural network inspired by Vaswani et al. (2017). The number of layers and attention heads are optimized during a hyper-parameter search, while the feed-forward layer dimensionality is fixed to 2,048. Input sequences of tokens are mapped to their corresponding embeddings, initialized by the embedding table of the tied-multi NMT model. Similarly to BERT (Devlin et al., 2019), a specific token is prepended to input sequence before being fed to the classifier. This token is finally fed during the forward pass to the output linear layer for sentence classification. The output linear layer has K dimensions, allowing to output as many logits as the number of layer combinations in the tied-multi NMT model. Finally, a sigmoid function outputs probabilities for each layer combination among the K possible combinations.

The parameters θ of the model f are learned using mini-batch stochastic gradient descent with Nesterov momentum (Sutskever et al., 2013) and the loss function $\mathcal{L}$, implemented as a weighted binary cross-entropy (BCE) function (3).

$$\mathcal{L}_{\text{BCE}^i_k} = -w^i_k \left[\delta_k y^i_k \cdot \log \hat{y}^i_k + (1 - y^i_k) \cdot \log(1 - \hat{y}^i_k)\right], \quad (3)$$

where y^i_k is the reference class of the i-th input sentence s^i, $\hat{y}^i_k$ is the output of the network after the sigmoid layer given s^i, and $\delta_k = (1 - p(t_k))^\alpha$ is the weight given to the k-th class based on class distribution prior. During our experiment, we have found that the classifier tends to favor recall in detriment to precision. To tackle this issue, we introduce another loss using an approximation of the macro F_β implemented following (4).

$$\mathcal{L}^i_{F_\beta} = 1 - \left[(1+\beta^2) \cdot \frac{P \cdot R}{(\beta^2 \cdot P) + R}\right], \quad (4)$$

where $P = \mu / \sum_k \hat{y}^i_k$, $R = \mu / \sum_k y^i_k$, and $\mu = \sum_k (\hat{y}^i_k \cdot y^i_k)$.

The final loss function is the linear interpolation of $\mathcal{L}_{\text{BCE}}$ averaged over the K classes and $\mathcal{L}_{F_\beta}$ with parameter λ, both averaged over the batch: $\lambda \times \mathcal{L}_{\text{BCE}} + (1-\lambda) \times \mathcal{L}_{F_\beta}$. We tune α, β, and λ during the classifier hyper-parameter search based on the validation loss.

4.2 Experiment

The layer combination classifier was trained on a subset of the training data for NMT models (Section 3.2) containing 5.00M sentences, whereas the remaining sentences compose a development and a test sets each containing approximately 200k sentences. The two latter subsets were used for hyper-parameter search and evaluation of the classifier, respectively. To allow for comparison and reproducibility, the final evaluation of the proposed approach in terms of translation quality and decoding speed were conducted on the official WMT development (newstest2017, 3,004 sentences) and test (newstest2018, 2,998 sentences) sets; the latter is the one also used in Section 3.2.

The training, development, and test sets were translated by each layer combination of the tied-multi NMT model. Each source sentence was thus aligned with 36 translations whose quality were measured by the chrF metric. Because several combinations can lead to the best score, the obtained dataset was labeled with multiple classes (36 layer combinations) and multiple labels (ties with regard to the metric). During inference, the ties were broken by selecting the layer combination with the highest value given by the sigmoid function, or backing-off to the deepest layer combination (6, 6) if no output value reaches 0.5. This tie breaking method differs from the oracle layer selection presented in Equation (1) and in Figure 2 which prioritizes shallowest layer combinations. In this experiment, decoding time was measured by processing one sentence at a time instead of batch decoding, the former being slower compared to the latter, but leads to precise results. The decoding times were 954s and 2,773s when using (1,1) and (6,6) layer combinations, respectively. By selecting the fastest encoder-decoder layer combinations

Classifier	Fine-tuning	Time (s)	BLEU
Baseline (tied (6,6))		2,773	35.0
Oracle (tied)		1,812	42.1
(#1) 8 layers, 8 heads	✓	2,736	35.0
(#2) 2 layers, 4 heads	✓	2,686	34.8
(#3) 2 layers, 4 heads		2,645	34.7
(#4) 4 layers, 2 heads		2,563	34.3

Table 2: Dynamic layer combination selection results in decoding time (in seconds, batch size of 1) and BLEU, including the baseline and oracle for the WMT newstest2018 using the tied-multi model architecture.

according to an oracle, the decoding times went down to 1,918s and 1,812s for the individual and tied-multi models, respectively. However, our objective is to be faster than default setting, that is, where one would choose (6,6) combination.

Several classifiers were trained and evaluated on the WMT test set, with or without fine-tuning on the WMT development set. Table 2 presents the results in terms of corpus-level BLEU and decoding speed.[12] Some classifiers maintain the translation quality (middle rows), whereas others show quality degradation but further gain in decoding speed (bottom rows). The classification results show that gains in decoding speed are possible with an a-priori decision for which encoder-decoder combination to select, based on the information contained in the source sentence only. However, no BLEU gain has so far been observed, demonstrating a trade-off between decoding speed and translation quality. Our best configuration for decoding speed (#4) reduced 210s but leads to a 0.7 point BLEU degradation. On the other hand, when preserving the translation quality compared to the baseline configuration (#1) we saved only 37s. The oracle layer combination can achieve substantial gains both in terms of BLEU (7.1 points) and decoding speed (961s). These oracle results motivate possible future work in layer combination prediction for the tied-multi NMT model.

5 Further Model Compression

We examined the combination of our multi-layer softmaxing approach with another parameter-tying method in neural networks, called *recurrent stacking* (RS) (Dabre and Fujita, 2019), complemented by *sequence-level knowledge distillation* (Kim and Rush, 2016), a specific type of knowledge distillation (Hinton et al., 2015). We demonstrate that these existing techniques help reduce the number of parameters in our model even further.

[12]Decoding time does not include the time spent for layer selection, which took up to 1.0 second for the entire test set.

5.1 Distillation into a Recurrently Stacked Model

In Section 2, we have discussed several model compression methods orthogonal to multi-layer softmaxing. Having already compressed $N \times M$ models with our approach, we consider further compressing it using RS. However, models that use RS layers tend to suffer from performance drops due to the large reduction in the number of parameters. As a way of compensating the performance drop, we apply sequence-level knowledge distillation.

First, we decode all source sentences in the training data to generate a pseudo-parallel corpus containing distillation target sentences, i.e., soft-targets for the child model which makes learning easier and hence is able to mimic the behavior of the parent model. Then, an RS child model is trained with multi-layer softmaxing on the generated pseudo-parallel corpus. Among a variety of distillation techniques, we chose the simplest one to show the impact that distillation can have in our setting, leaving an extensive exploration of more complex methods for the future.

5.2 Experiment

We conducted an experiment to show that RS and sequence distillation can lead to an extremely compressed tied-multi model which no longer suffers from performance drops. We compared the following four variations of our tied-multi model trained with multi-layer softmaxing.

Tied-multi model: A model that does not share the parameters across layers, trained on the original parallel corpus.

Distilled tied-multi model: The same model as above but trained on the pseudo-parallel corpus.

Tied-multi RS model: A tied-multi model that uses RS layers, trained on the original parallel corpus.

Distilled tied-multi RS model: The same model as above but trained on the pseudo-parallel corpus.

First, we trained 5 vanilla models with 6 encoder and 6 decoder layers, because the performance of

		Tied-multi model						Tied-multi RS model					
	$n \backslash m$	1	2	3	4	5	6	1	2	3	4	5	6
without distillation	1	23.2	28.6	30.5	30.8	31.2	31.5	25.7	29.8	30.6	30.8	30.7	30.9
	2	26.5	31.5	33.0	33.6	33.8	34.0	28.5	32.3	32.9	33.0	33.1	33.2
	3	27.8	32.5	33.9	34.6	34.7	34.7	29.2	32.9	33.6	33.8	33.6	33.5
	4	28.3	33.0	34.3	34.8	34.9	34.9	29.3	33.2	33.7	33.9	33.6	33.7
	5	28.6	33.1	34.5	34.8	35.0	35.1	29.4	33.2	33.7	33.9	33.9	34.0
	6	28.7	33.1	34.6	34.7	34.9	35.0	29.2	33.2	33.7	33.9	34.0	33.8
with distillation	1	30.1	34.0	35.1	35.3	35.6	35.7	31.2	33.5	34.1	34.2	34.3	34.3
	2	33.4	35.8	36.6	36.8	37.1	37.3	33.7	35.5	35.7	35.7	35.8	35.8
	3	34.7	36.5	37.0	37.4	37.4	37.5	34.1	35.8	36.1	36.1	36.2	36.2
	4	35.2	36.8	37.2	37.4	37.5	37.5	34.3	36.0	36.2	36.2	36.3	36.3
	5	35.5	36.9	37.1	37.4	37.5	37.6	34.5	36.1	36.2	36.3	36.3	36.3
	6	35.5	37.0	37.2	37.5	37.6	37.6	34.6	36.1	36.2	36.2	36.3	36.2

Table 3: BLEU scores of the tied-multi models with (left block) and without (center and right blocks) RS layers, each trained with (bottom block) and without (top block) sequence distillation. n and m respectively denote the number of layers in the encoder and the decoder. The top-left block is identical to the middle block in Table 1.

distilled models is affected by the quality of parent models, and NMT models vary vastly in performance (around 2.0 BLEU points) depending on parameter initialization. We then decode the source side (English side) of the entire training data (5.58M sentences) with the one[13] with the highest BLEU score on the newstest2017 (used in Section 4.2) in order to generate pseudo-parallel corpus for sequence distillation.

Table 3 gives the BLEU scores for all models. Comparing top-left and top-right blocks of the table, we can see that the BLEU scores for RS models are higher than their non-RS counterparts when using fewer than 3 decoder layers. This shows the benefit of RS layers despite the large parameter reduction. However, the reduction in parameters negatively affects (up to 1.3 BLEU points) when decoding with more decoder layers, confirming the limitation of RS as expected.

Comparing the scores of the top and bottom halves of the table, we can see that distillation dramatically boosts the performance of the shallower encoder and decoder layers. For instance, without distillation, the tied-multi model gave a BLEU of 23.2 when decoding with 1 encoder and 1 decoder layers, but the same layer combination reaches 30.1 BLEU through distillation. Given that RS further improves performance using lower layers, the BLEU score increases to 31.2. As such, distillation enables decoding using fewer layers without substantial drops in performance. Furthermore, the BLEU scores did not vary significantly when the layers deeper than 3 were used, meaning that we might as well train shallower models using distillation. The performance of our final model, i.e., the distilled tied-multi RS model (bottom-right), was significantly lower than the non-RS model (up to 1.5 BLEU points) similarly to its non-distilled counterpart. However, given that it outperforms our original tied-multi model (top-left) in all the encoder-decoder layer combinations, we conclude that we can obtain a highly compact model with better performance.

[13]Ensemble of multiple models (Freitag et al., 2017) is commonly used for distillation, but we used a single model to save decoding time.

Model(s)	Parameters	Relative size
36 vanilla models	4,608M	25.16
Single tied-multi model	183M	1.00
36 RS models	2,623M	14.33
Single tied-multi RS model	73M	0.40

Table 4: Total model sizes for covering all 36 encoder-decoder layer combinations. The relative size is calculated regarding the tied-multi model as a standard. Similarly to "36 vanilla models," "36 RS models" represents the total number of parameters of all models.

We now analyze the effect of RS and knowledge distillation on model size and decoding speed.

Model Size: Table 4 gives the sizes of several models and their ratio with respect to the tied-multi model. Training vanilla and RS models with 36 different encoder-decoder layer combinations required 25.2 and 14.3 times the number of parameters of a single tied-multi model, respectively. Although RS led to some parameter reduction, combining RS with our tied-multi model resulted in a further compressed single model. This model has 63.2 times and 36.0 times fewer parameters than all the individual vanilla and RS models, respectively.

Decoding Speed: Table 5 compares results obtained by beam and greedy search using our distilled tied-multi RS model. In general, greedy

		BLEU score						Decoding time (sec)					
	$n\backslash m$	1	2	3	4	5	6	1	2	3	4	5	6
Beam search	1	31.2	33.5	34.1	34.2	34.3	34.3	94.7	101.9	143.4	174.7	215.5	244.5
	2	33.7	35.5	35.7	35.7	35.8	35.8	100.5	110.8	153.7	185.6	227.8	253.6
	3	34.1	35.8	36.1	36.1	36.2	36.2	102.5	114.2	168.5	194.8	234.0	259.8
	4	34.3	36.0	36.2	36.2	36.3	36.3	104.1	105.6	143.9	197.0	219.1	264.6
	5	34.5	36.1	36.2	36.3	36.3	36.3	105.1	111.5	156.4	186.0	236.1	268.8
	6	34.6	36.1	36.2	36.2	36.3	36.2	107.4	113.6	168.1	190.1	229.5	257.9
Greedy search	1	30.4	33.1	33.8	34.0	33.8	33.9	58.8	69.1	78.4	94.6	110.7	124.3
	2	33.2	35.0	35.3	35.5	35.4	35.5	62.7	68.0	78.8	94.1	112.8	125.8
	3	33.8	35.5	35.7	35.7	35.8	35.8	71.8	70.3	79.3	99.9	114.9	128.4
	4	34.0	35.8	35.8	35.8	35.8	35.8	72.1	70.7	82.3	98.8	115.2	127.5
	5	34.0	35.7	35.8	35.8	35.8	35.9	76.2	68.6	80.9	99.4	120.9	136.7
	6	34.1	35.6	35.8	35.8	35.8	35.9	76.6	69.3	81.5	99.2	120.7	131.5

Table 5: BLEU scores and decoding times of our distilled tied-multi RS model by beam and greedy search. The top-left block is identical to the bottom-right block in Table 3. The top-right block is identical to the right-most block in Table 1.

decoding is faster than beam decoding, but suffers from reduced performance. By using our distilled model, however, greedy decoding reduced the BLEU scores only by 0.5 points compared to beam decoding. For instance, whereas beam decoding with our tied-multi model without RS and distillation (top-left block in Table 3) achieved the highest BLEU score of 35.1 with 5 encoder and 6 decoder layers consuming 268.8s, greedy decoding with our distilled tied-multi RS model with 2 encoder and 2 decoder layers resulted in a comparable BLEU score of 35.0 in 68.0s, i.e., with a factor of 4.0 in decoding time thanks to RS and distillation. This happens because we have used translations generated by beam decoding as target sentences for knowledge distillation, which has the ability to loosely distill beam search behavior into greedy decoding behavior (Kim and Rush, 2016).

6 Conclusion

In this paper, we have proposed a novel procedure for training encoder-decoder models, where the softmax function is applied to the output of each of the M decoder layers derived using the output of each of the N encoder layers. This compresses $N \times M$ models into a single model that can be used for decoding with a variable number of encoder ($1 \leq n \leq N$) and decoder ($1 \leq m \leq M$) layers. This model can be used in different latency scenarios and hence is highly versatile. We have made a cost-benefit analysis of our method, taking NMT as a case study of encoder-decoder models. We have proposed and evaluated two orthogonal extensions and show that we can (a) dynamically choose layer combinations for slightly faster decoding and (b) further compress models using recurrent stacking with knowledge distillation leading to models that also enable faster decoding.

For further speed up in decoding as well as model compression, we plan to combine our approach with other techniques, such as those mentioned in Section 2. Although we have only tested our idea for NMT, it should be applicable to other tasks based on deep neural networks.

Acknowledgments

We would like to thank all the reviewers for their insightful comments and suggestions. A part of this work was conducted under the program “Research and Development of Enhanced Multilingual and Multipurpose Speech Translation System” of the Ministry of Internal Affairs and Communications (MIC), Japan. Atsushi Fujita was partly supported by JSPS KAKENHI Grant Number 19H05660.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations*, San Diego, USA.

Yonatan Belinkov, Lluís Màrquez, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. 2017. Evaluating layers of representation in neural machine translation on part-of-speech and semantic tagging tasks. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1–10, Taipei, Taiwan.

Kyunghyun Cho, Bart van Merriënboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, Doha, Qatar.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830.

Raj Dabre and Atsushi Fujita. 2019. Recurrent stacking of layers for compact neural machine translation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6292–6299, Honolulu, USA.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional Transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, USA.

Zi-Yi Dou, Zhaopeng Tu, Xing Wang, Shuming Shi, and Tong Zhang. 2018. Exploiting deep representations for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4253–4262, Brussels, Belgium.

Markus Freitag, Yaser Al-Onaizan, and Baskaran Sankaran. 2017. Ensemble distillation for neural machine translation. *CoRR*, abs/1702.01802.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1737–1746, Lille, France.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, USA.

Shankar Kumar and William Byrne. 2004. Minimum Bayes-risk decoding for statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 169–176, Boston, USA.

Darryl D. Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, pages 2849–2858, New York, USA.

Qingsong Ma, Ondřej Bojar, and Yvette Graham. 2018. Results of the WMT18 metrics shared task. In *Proceedings of the Third Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 682–701, Brussels, Belgium.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9, Brussels, Belgium.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318, Philadelphia, USA.

Maja Popović. 2016. chrF deconstructed: β parameters and n-gram weights. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 499–504, Berlin, Germany.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium.

Abigail See, Minh-Thang Luong, and Christopher D. Manning. 2016. Compression of neural machine translation models via pruning. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 291–301, Berlin, Germany.

Lucia Specia, Dhwaj Raj, and Marco Turchi. 2010. Machine translation evaluation versus quality estimation. *Machine Translation*, 24(1):39–50.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *Proceedings of the International Conference on Machine Learning*, pages 1139–1147, Atlanta, USA.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th Neural Information Processing Systems Conference*, pages 3104–3112, Montréal, Canada.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 30th Neural Information Processing Systems Conference*, pages 5998–6008, Long Beach, USA.

Qiang Wang, Fuxue Li, Tong Xiao, Yanyang Li, Yinqiao Li, and Jingbo Zhu. 2018. Multi-layer representation fusion for neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3015–3026, Santa Fe, USA.

Yingce Xia, Tianyu He, Xu Tan, Fei Tian, Di He, and Tao Qin. 2019. Tied Transformers: Neural machine translation with shared encoder and decoder. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5466–5473, Honolulu, USA.

Deyi Xiong, Biao Zhang, and Jinsong Su. 2018. Accelerating neural Transformer via an average attention network. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Long Papers*, pages 1789–1798, Melbourne, Australia.